An (almost) Easy INTRODUCTION TO OPENACC®

Carlos J. Barrios H., PhD
@carlosjaimebh
Acknowledgements

- Sunita Chandrasekaran, University of Delaware
- Guido Juckeland, Helmholtz-Zentrum Dresden-Rossendorf (HZDR)
- Fernanda Foertter, Oak Ridge National Laboratory
- Joe Bongo, NVIDIA Deep Learning Institute
GPU Computing is Powerful…

… but it’s not simple.
(REMEMBER) INTRODUCTION TO PARALLEL PROGRAMMING
Remember: WHAT IS PARALLEL PROGRAMMING?

- “Performance Programming”
- Parallel programming involves exposing an algorithm’s ability to execute in parallel
- This may involve breaking a large operation into smaller tasks (task parallelism)
- Or doing the same operation on multiple data elements (data parallelism)
- Parallel execution enables better performance on modern hardware
A REAL WORLD CASE STUDY

Modern cancer research

- The Russian Academy of Science created a program to simulate light propagation through human tissue
- This program was used to be able to more accurately detect cancerous cells by simulating billions of random paths that the light could take through human tissue
- With parallel programming, they were able to run thousands of these paths simultaneously
- The sequential program took 2.5 hours to run
- The parallel version took less than 2 minutes

Parallel Computing Illuminating a Path to Early Cancer Detection
WHAT IS PARALLEL PROGRAMMING?

A real world example

- A professor and his 3 teaching assistants (TA) are grading 1,000 student exams
- This exam has 8 questions on it
- Let’s assume it takes 1 minute to grade 1 question on 1 exam
- To maintain fairness, if someone grades a question (for example, question #1) then they must grade that question on all other exams
- The following is a sequential version of exam grading
SEQUENTIAL SOLUTION

Grade Exams 1-1000: Questions #1, 2, 3, 4, 5, 6, 7, 8 - 8000m

Prof

TA 1

TA 2

TA 3

8000 m
SEQUENTIAL SOLUTION

Exams 1-1000
: Q #1 : 1000m

Exams 1-1000
: Q #2 : 1000m

Exams 1-1000
: Q #3 : 1000m

Exams 1-1000
: Q #4 : 1000m

Exams 1-1000
: Q #5 : 1000m

Exams 1-1000
: Q #6 : 1000m

Exams 1-1000
: Q #7 : 1000m

Exams 1-1000
: Q #8 : 1000m

8000+ m
SEQUENTIAL SOLUTION

Exams 1-1000: Q #1, 2: 2000m

Exams 1-1000: Q #3, 4: 2000m

Exams 1-1000: Q #5, 6: 2000m

Exams 1-1000: Q #7, 8: 2000m

8000+ m
PARALLEL SOLUTION

Prof
- Exams 1-250: Q #1, 2: 500m
- Exams 751-1000: Q #1, 2: 500m
- Exams 501-750: Q #1, 2: 500m
- Exams 251-500: Q #3, 4: 500m
- Exams 501-750: Q #3, 4: 500m
- Exams 251-500: Q #3, 4: 500m
- Exams 751-1000: Q #1, 2: 500m
- Exams 501-750: Q #1, 2: 500m
- Exams 251-500: Q #3, 4: 500m
- Exams 751-1000: Q #3, 4: 500m

TA 1
- Exams 1-250: Q #3, 4: 500m
- Exams 251-500: Q #3, 4: 500m
- Exams 501-750: Q #1, 2: 500m
- Exams 251-500: Q #5, 6: 500m
- Exams 751-1000: Q #5, 6: 500m
- Exams 501-750: Q #5, 6: 500m
- Exams 251-500: Q #5, 6: 500m
- Exams 751-1000: Q #5, 6: 500m

TA 2
- Exams 251-500: Q #5, 6: 500m
- Exams 501-750: Q #5, 6: 500m
- Exams 251-500: Q #1, 2: 500m
- Exams 751-1000: Q #1, 2: 500m
- Exams 251-500: Q #7, 8: 500m
- Exams 751-1000: Q #7, 8: 500m
- Exams 501-750: Q #7, 8: 500m
- Exams 251-500: Q #7, 8: 500m

TA 3
- Exams 501-750: Q #7, 8: 500m
- Exams 251-500: Q #7, 8: 500m
- Exams 751-1000: Q #7, 8: 500m
- Exams 501-750: Q #7, 8: 500m
- Exams 251-500: Q #3, 4: 500m
- Exams 751-1000: Q #3, 4: 500m
- Exams 501-750: Q #3, 4: 500m
- Exams 251-500: Q #3, 4: 500m

2000+ m
PIPELINE STALL

Q #1, 2 2m
Q #1, 2 2m
Q #1, 2 2m
Q #1, 2 2m
Q #3, 4 2m
Q #3, 4 2m
Q #3, 4 2m
Q #3, 4 2m
Q #5, 6 2m
Q #5, 6 2m
Q #5, 6 2m
Q #5, 6 2m
Q #7, 8 2m
Q #7, 8 2m
Q #7, 8 2m
Q #7, 8 2m

2006+ m
GRADING EXAMPLE SUMMARY

It’s critical to understand the problem before trying to parallelize it

- Can the work be done in an arbitrary order, or must it be done in sequential order?

- Does each task take the same amount of time to complete? If not, it may be necessary to “load balance.”

In our example, the only restriction is that a single question be graded by a single grader, so we could divide the work easily, but had to communicate periodically.

- This case study is an example of task-based parallelism. Each grader is assigned a task like “Grade questions 1 & 2 on the first 500 tests”

- If instead each question could be graded by different graders, then we could have data parallelism: all graders work on Q1 of the following tests, then Q2, etc.
(Remember) AMDAHL’S LAW
Amdahl’s law is an observation that how much speed-up you get from parallelizing the code is limited by the remaining serial part.

Any remaining serial code will reduce the possible speed-up.

This is why it’s important to focus on parallelizing the most time consuming parts, not just the easiest.
APPLYING AMDAHL’S LAW

Estimating Potential Speed-up

- What’s the maximum speed-up that can be obtained by parallelizing 50% of the code?
  
  \[(1 / 100\% - 50\%) = (1 / 1.0 - 0.50 ) = 2.0X\]

- What’s the maximum speed-up that can be obtained by parallelizing 25% of the code?
  
  \[(1 / 100\% - 25\%) = (1 / 1.0 - 0.25 ) = 1.3X\]

- What’s the maximum speed-up that can be obtained by parallelizing 90% of the code?
  
  \[(1 / 100\% - 90\%) = (1 / 1.0 - 0.90 ) = 10.0X\]
(NOW) AN INTRODUCTION TO OPENACC
OpenACC
Simple | Powerful | Portable
Fueling the Next Wave of Scientific Discoveries in HPC

main()
{
    <serial code>
    #pragma acc kernels
    // automatically runs on GPU
    {
        <parallel code>
    }
}

RIKEN Japan
NICAM- Climate Modeling
7-8x Speed-Up
5% of Code Modified

University of Illinois
PowerGrid- MRI Reconstruction
70x Speed-Up
2 Days of Effort

8000+
Developers
using OpenACC

http://www.cray.com/sites/default/files/resources/OpenACC_213462_12_OpenACC_Cosmo_CS_FNL.pdf
http://www.openacc.org/content/experiences-porting-molecular-dynamics-code-gpus-cray-xk7
OpenACC makes GPU computing approachable for domain scientists. Initial OpenACC implementation required only minor effort, and more importantly, *no modifications* of our existing CPU implementation.

Janus Juul Eriksen, PhD Fellow
qLEAP Center for Theoretical Chemistry, Aarhus University
OpenACC Directives

- Manage Data Movement
- Initiate Parallel Execution
- Optimize Loop Mappings

```c
#pragma acc data copyin(a,b) copyout(c)
{
    ...
    #pragma acc parallel
    {
        #pragma acc loop gang vector
        for (i = 0; i < n; ++i) {
            z[i] = x[i] + y[i];
            ...
        }
    }
    ...
}
```

- Incremental
- Single source
- Interoperable
- Performance portable
- CPU, GPU, MIC
OpenACC is designed to be portable to many existing and future parallel platforms. The programmer need not think about specific hardware details, but rather express the parallelism in generic terms. An OpenACC program runs on a *host* (typically a CPU) that manages one or more parallel *devices* (GPUs, etc.). The host and device(s) are logically thought of as having separate memories.
OPENACC

Three major strengths

- Incremental
- Single Source
- Low Learning Curve
**OPENACC**

**Incremental**

- Maintain existing sequential code
- Add annotations to expose parallelism
- After verifying correctness, annotate more of the code

```
#pragma acc parallel loop
for( i = 0; i < N; i++ )
{
    < loop code >
}

#pragma acc parallel loop
for( i = 0; i < N; i++ )
{
    < loop code >
}
```

Begin with a working sequential code.
Parallelize it with OpenACC.
Rerun the code to verify correct behavior, remove/alter OpenACC code as needed.
OPENACC

- Incremental
  - Maintain existing sequential code
  - Add annotations to expose parallelism
  - After verifying correctness, annotate more of the code

- Single Source

- Low Learning Curve
OPENACC

**Supported Platforms**

- POWER
- Sunway
- x86 CPU
- x86 Xeon Phi
- NVIDIA GPU
- PEZY-SC

**Single Source**

- Rebuild the same code on multiple architectures
- Compiler determines how to parallelize for the desired machine
- Sequential code is maintained

The compiler can **ignore** your OpenACC code additions, so the same code can be used for **parallel** or **sequential** execution.

```c
int main()
{
    // ...
    #pragma acc parallel loop
    for(int i = 0; i < N; i++)
        < loop code>
}
```
OPENACC

Incremental
- Maintain existing sequential code
- Add annotations to expose parallelism
- After verifying correctness, annotate more of the code

Single Source
- Rebuild the same code on multiple architectures
- Compiler determines how to parallelize for the desired machine
- Sequential code is maintained

Low Learning Curve
OPENACC

- OpenACC is meant to be easy to use, and easy to learn
- Programmer remains in familiar C, C++, or Fortran
- No reason to learn low-level details of the hardware.

```c
int main(){
    <sequential code>
    #pragma acc kernels
    {
        <parallel code>
    }
}
```

The programmer will give hints to the compiler about which parts of the code to parallelize. The compiler will then generate parallelism for the target parallel hardware.
OPENACC

**Incremental**
- Maintain existing sequential code
- Add annotations to expose parallelism
- After verifying correctness, annotate more of the code

**Single Source**
- Rebuild the same code on multiple architectures
- Compiler determines how to parallelize for the desired machine
- Sequential code is maintained

**Low Learning Curve**
- OpenACC is meant to be easy to use, and easy to learn
- Programmer remains in familiar C, C++, or Fortran
- No reason to learn low-level details of the hardware.
OPENACC RESOURCES

Guides ● Talks ● Tutorials ● Videos ● Books ● Spec ● Code Samples ● Teaching Materials ● Events ● Success Stories ● Courses ● Slack ● Stack Overflow

FREE Compilers

https://www.openacc.org/community#slack

Resources
https://www.openacc.org/resources

Success Stories
https://www.openacc.org/success-stories

Compilers and Tools
https://www.openacc.org/tools

Events
https://www.openacc.org/events

https://www.openacc.org/resources

https://www.openacc.org/success-stories

https://www.openacc.org/tools

https://www.openacc.org/events
OpenACC Programming Cycle
Example: Jacobi Iteration

Iteratively converges to correct value (e.g. Temperature), by computing new values at each point from the average of neighboring points.

Common, useful algorithm

Example: Solve Laplace equation in 2D: $\nabla^2 f(x, y) = 0$
while ( err > tol && iter < iter_max ) { 
    err=0.0;

    for( int j = 1; j < n-1; j++ ) {
        for( int i = 1; i < m-1; i++ ) {
            err = max(err, abs(Anew[j][i] - A[j][i]));
        }
    }

    for ( int j = 1; j < n-1; j++) {
        for( int i = 1; i < m-1; i++ ) {
            A[j][i] = Anew[j][i];
        }
    }
    iter++;
}
Identify Available Parallelism

Optimize Loop Performance

Express Parallelism

Express Data Movement
Identify Parallelism

```c
while ( err > tol && iter < iter_max ) {
    err=0.0;

    for( int j = 1; j < n-1; j++ ) {
        for( int i = 1; i < m-1; i++ ) {
            err = max(err, abs(Anew[j][i] - A[j][i]));
        }
    }

    iter++;
}
```
The kernels directive identifies a region that may contain *loops* that the compiler can turn into parallel *kernels*.

```c
#pragma acc kernels
{
    for(int i=0; i<N; i++)
    {
        x[i] = 1.0;
        y[i] = 2.0;
    }

    for(int i=0; i<N; i++)
    {
        y[i] = a*x[i] + y[i];
    }
}
```

The compiler identifies 2 parallel loops and generates 2 kernels.
Parallelize with OpenACC kernels

while ( err > tol && iter < iter_max ) {
    err=0.0;

    #pragma acc kernels
    {
        for( int j = 1; j < n-1; j++ ) {
            for(int i = 1; i < m-1; i++ ) {


                err = max(err, abs(Anew[j][i] - A[j][i]));
            }
        }

        for( int j = 1; j < n-1; j++ ) {
            for( int i = 1; i < m-1; i++ ) {
                A[j][i] = Anew[j][i];
            }
        }

        iter++;
    }
Building the code

$ pgcc -fast -ta=tesla -Minfo=all laplace2d.c
main:
  40, Loop not fused: function call before adjacent loop
  Generated vector sse code for the loop
  51, Loop not vectorized/parallelized: potential early exits
  55, Generating copyout(Anew[1:4094][1:4094])
      Generating copyin(A[::][::])
      Generating copyout(A[1:4094][1:4094])
      Generating Tesla code
  57, Loop is parallelizable
  59, Loop is parallelizable
      Accelerator kernel generated
      57, #pragma acc loop gang /* blockIdx.y */
      59, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
  63, Max reduction generated for error
  67, Loop is parallelizable
  69, Loop is parallelizable
      Accelerator kernel generated
      67, #pragma acc loop gang /* blockIdx.y */
      69, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
Why did OpenACC slow down here?

Intel Xeon E5-2698 v3 @ 2.30GHz (Haswell) vs. NVIDIA Tesla K40
PCIe Copies

104ms/iteration
while ( err > tol && iter < iter_max )
{
    err=0.0;
    #pragma acc kernels
    for( int j = 1; j < n-1; j++) {
        for(int i = 1; i < m-1; i++) {
            err = max(err, abs(Anew[j][i] - A[j][i]));
        }
    }
    ...
while ( err > tol && iter < iter_max ) {
    err=0.0;

#pragma acc kernels
{
    for( int j = 1; j < n-1; j++ ) {
        for(int i = 1; i < m-1; i++) {
            err = max(err, abs(Anew[j][i] - A[j][i]));
        }
    }
}

for( int j = 1; j < n-1; j++ ) {
    for( int i = 1; i < m-1; i++ ) {
        A[j][i] = Anew[j][i];
    }
}

iter++;
}
Identify Available Parallelism

Express Parallelism

Optimize Loop Performance

Express Data Movement
Data regions

The data directive defines a region of code in which GPU arrays remain on the GPU and are shared among all kernels in that region.

```
#pragma acc data
{
#pragma acc kernels
...

#pragma acc kernels
...
}
```

Arrays used within the data region will remain on the GPU until the end of the data region.
Data Clauses

**copy** \((\text{list})\)  
Allocates memory on GPU and copies data from host to GPU when entering region and copies data to the host when exiting region.

**copyin** \((\text{list})\)  
Allocates memory on GPU and copies data from host to GPU when entering region.

**copyout** \((\text{list})\)  
Allocates memory on GPU and copies data to the host when exiting region.

**create** \((\text{list})\)  
Allocates memory on GPU but does not copy.

**present** \((\text{list})\)  
Data is already present on GPU from another containing data region.

**deviceptr** \((\text{list})\)  
The variable is a device pointer (e.g. CUDA) and can be used directly on the device.
Array Shaping

Compiler sometimes cannot determine size of arrays

Must specify explicitly using data clauses and array “shape”

C/C++

#pragma acc data copyin(a[0:nelem]) copyout(b[s/4:3*s/4])

Fortran

!$acc data copyin(a(1:end)) copyout(b(s/4:3*s/4))

Note: data clauses can be used on data, parallel, or kernels
#pragma acc data copy(A) create(Anew)
while (err > tol && iter < iter_max) {
    err=0.0;
#pragma acc kernels
{
    for(int j = 1; j < n-1; j++) {
        for(int i = 1; i < m-1; i++) {

            Anew[j][i] = 0.25 * (A[j][i+1] + A[j][i-1] +
                                   A[j-1][i] + A[j+1][i]);

            err = max(err, abs(Anew[j][i] - A[j][i]));
        }
    }

    for(int j = 1; j < n-1; j++) {
        for(int i = 1; i < m-1; i++ ) {
            A[j][i] = Anew[j][i];
        }
    }
} iter++;} iter++;

Copy A to/from the accelerator only when needed.
Create Anew as a device temporary.
Rebuilding the code

$ pgcc -fast -acc -ta=tesla -Minfo=all laplace2d.c

main:
  40, Loop not fused: function call before adjacent loop
  Generated vector sse code for the loop
  51, Generating copy(A[::][::])
      Generating create(Anew[::][::])
      Loop not vectorized/parallelized: potential early exits
  56, Accelerator kernel generated
      56, Max reduction generated for error
      57, #pragma acc loop gang /* blockIdx.x */
          59, #pragma acc loop vector(256) /* threadIdx.x */
  56, Generating Tesla code
  59, Loop is parallelizable
  67, Accelerator kernel generated
      68, #pragma acc loop gang /* blockIdx.x */
          70, #pragma acc loop vector(256) /* threadIdx.x */
  67, Generating Tesla code
  70, Loop is parallelizable
Visual Profiler: Data Region

Was 104ms
Speed-Up (Higher is Better)

Intel Xeon E5-2698 v3 @ 2.30GHz (Haswell) vs. NVIDIA Tesla K40

- Single Thread: 1.90X
- 2 Threads: 1.90X
- 4 Threads: 3.20X
- 6 Threads: 3.74X
- 8 Threads: 3.83X
- OpenACC: 19.89X

Socket/Socket: 5.2X
Identify Available Parallelism

Express Parallelism

Optimize Loop Performance

Express Data Movement
The loop Directive

The `loop` directive gives the compiler additional information about the next loop in the source code through several clauses.

- **independent** - all iterations of the loop are independent
- **collapse(N)** - turn the next N loops into one, flattened loop
- **tile(N[,M,...])** - break the next 1 or more loops into tiles based on the provided dimensions.

These clauses and more will be discussed in greater detail in a later class.
Optimize Loop Performance

```c
#pragma acc data copy(A) create(Anew)
while ( err > tol && iter < iter_max ) {
    err=0.0;
    #pragma acc kernels
    {
        #pragma acc loop device_type(nvidia) tile(32,4) for( int j = 1; j < n-1; j++) {
            for(int i = 1; i < m-1; i++) {
                err = max(err, abs(Anew[j][i] - A[j][i]));
            }
        }
        #pragma acc loop device_type(nvidia) tile(32,4) for( int j = 1; j < n-1; j++) {
            for( int i = 1; i < m-1; i++) {
                A[j][i] = Anew[j][i];
            }
        }
    }
    iter++;
}
```

“Tile” the next two loops into 32x4 blocks, but only on NVIDIA GPUs.
Speed-Up (Higher is Better)

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Speed-Up</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Thread</td>
<td>0.00X</td>
</tr>
<tr>
<td>2 Threads</td>
<td>1.90X</td>
</tr>
<tr>
<td>4 Threads</td>
<td>3.20X</td>
</tr>
<tr>
<td>6 Threads</td>
<td>3.74X</td>
</tr>
<tr>
<td>8 Threads</td>
<td>3.83X</td>
</tr>
<tr>
<td>OpenACC</td>
<td>19.89X</td>
</tr>
<tr>
<td>OpenACC Tuned</td>
<td>21.22X</td>
</tr>
</tbody>
</table>

Intel Xeon E5-2698 v3 @ 2.30GHz (Haswell) vs. NVIDIA Tesla K40
The OpenACC Toolkit
Introducing the New OpenACC Toolkit
Free Toolkit Offers Simple & Powerful Path to Accelerated Computing

PGI Compiler
Free OpenACC compiler for academia

NVProf Profiler
Easily find where to add compiler directives

GPU Wizard
Identify which GPU libraries can jumpstart code

Code Samples
Learn from examples of real-world algorithms

Documentation
Quick start guide, Best practices, Forums

http://developer.nvidia.com/openacc
Download the OpenACC Toolkit

Go to https://developer.nvidia.com/openacc
Download the OpenACC Toolkit

- Go to https://developer.nvidia.com/openacc

- Register for the toolkit
  - If you are an academic developer, be sure to click the check box at the bottom.
Download the OpenACC Toolkit

- Go to https://developer.nvidia.com/openacc
- Register for the toolkit
  - If you are an academic developer, be sure to click the check box at the bottom.
- You will receive an email from NVIDIA
  - Be sure to read the Quick Start Guide
Windows/Mac Developers

• The OpenACC Toolkit is only available on Linux, however...

• The PGI compiler is available on Mac and Windows from http://www.pgroup.com/support/trial.htm
  • You should still register for the OpenACC Toolkit to get the 90 day license.

• The CUDA Toolkit contains the libraries and profiling tools that will be used in this course.
  • https://developer.nvidia.com/cuda-zone

  • Obtaining all examples and guides from the toolkit will still require downloading the full OpenACC toolkit.
Install the OpenACC Toolkit

- Go to developer.nvidia.com/openacc
- Register for the OpenACC Toolkit
- Install on your personal machine. (Linux Only)
Where to find help

• OpenACC Course Recordings - https://developer.nvidia.com/openacc-course
• OpenACC on StackOverflow - http://stackoverflow.com/questions/tagged/openacc
• OpenACC Toolkit - http://developer.nvidia.com/openacc

Additional Resources:

• Parallel Forall Blog - http://devblogs.nvidia.com/parallelforall/
• GPU Technology Conference - http://www.gputechconf.com/
• OpenACC Website - http://openacc.org/
Get Started

OpenACC is a user-driven directive-based performance-portable parallel programming model. It is designed for scientists and engineers interested in porting their codes to a wide-variety of heterogeneous HPC hardware platforms and architectures with significantly less programming effort than required with a low-level model.

Then, follow the exercises of the OpenAcc tutorial: https://www.openacc.org/get-started
THANK YOU

@carlosjaimebh